Adaptive Granularity Memory Systems: A Tradeoff between Storage Efficiency and Throughput

Doe Hyun Yoon, Min Kyu Jeong, and Mattan Erez
The University of Texas at Austin
Electrical and Computer Engineering Dept.
doehyun.yoon@gmail.com, mjeong@ece.utexas.edu, mattan.erez@mail.utexas.edu

ABSTRACT

We propose *adaptive granularity* to combine the best of fine-grained and coarse-grained memory accesses. We augment virtual memory to allow each page to specify its preferred granularity of access based on spatial locality and error-tolerance tradeoffs. We use sector caches and sub-ranked memory systems to implement adaptive granularity. We also show how to incorporate adaptive granularity into memory access scheduling. We evaluate our architecture with and without ECC using memory intensive benchmarks from the SPEC, Olden, PARSEC, SPLASH2, and HPCS benchmark suites and micro-benchmarks. The evaluation shows that performance is improved by 61% without ECC and 44% with ECC in memory-intensive applications, while the reduction in memory power consumption (29% without ECC and 14% with ECC) and traffic (78% without ECC and 66% with ECC) is significant.

Categories and Subject Descriptors

B.3.1 [**Memory Structure**]: Semiconductor Memories—*Dynamic memory (DRAM)*; B.3.4 [**Memory Structure**]: Reliability, Testing, and Fault-Tolerance—*Error-Checking*

General Terms

Design, Performance, Reliability

Keywords

Access granularity, ECC, Main memory

1. INTRODUCTION

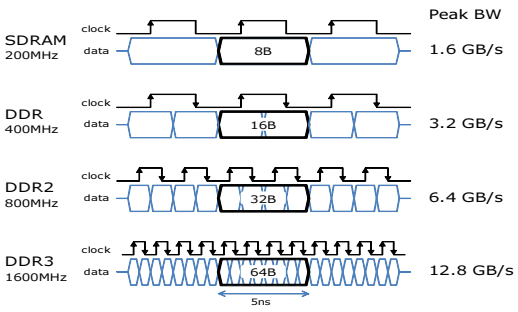
The continuing improvements in device density and in the potential performance of parallel processors place increased pressure on the throughput, power consumption, and reliability. The fundamental problem is that systems require reliable high-capacity memory with high throughput and low power all at the same time, while these parameters are often in conflict.

Current systems generally try to achieve the above goals by tuning for mostly sequential and coarse-grained memory accesses. Coarse-grained accesses allow efficient use of modern DDRx DRAMs and also enable effective low-redundancy error tolerance [14]. Unfortunately, when spatial locality is low, the coarse-grained approach is very inefficient in terms of power and throughput. The alternative of using fine-grained access is simply too expensive for most systems, although it is used in some very high-end vector processors [6]. We present an architecture with dynamically-tunable and adaptive memory access granularity. Our work enables the processor to selectively use fine-grained access, only when beneficial and still maintain the efficiency of coarse-grained access by default. We explore the tradeoffs involved in designing this *adaptive granularity memory system* (AGMS), including issues relating to error tolerance and memory access scheduling.

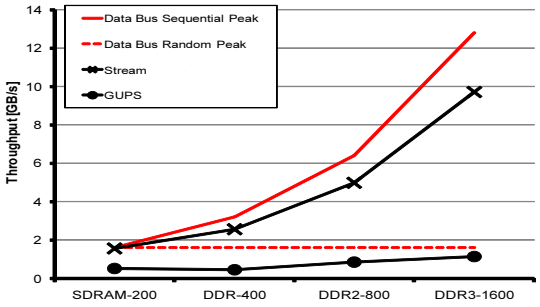
Prior work showed that many applications have a large fraction of data that has low spatial locality because of non-unit strides, indexed gather/scatter access, and other complex access patterns [36, 34, 41]. Research on mitigating the negative impact of low spatial locality on cache-based coarse-grained access systems, however, focused exclusively on improving cache-array utilization [28, 36]. Our observation is that improving the efficiency of the cache array is no longer sufficient and that for many applications, memory throughput is the bottleneck, and memory power consumption is significant. Memory systems that only make fine-grained accesses can work well when gather/scatter accesses are dominant, but squander the benefits of coarse-grained accesses in the common case of high spatial locality. As a result, fine-grain-only memory systems are a poor choice for most systems from the perspective of both performance and cost, especially when a high level of memory reliability and availability is required. As we discuss in Section 2, a fine-grained access requires much more redundancy to ensure correct execution, reducing effective capacity, bandwidth, and power efficiency. Therefore, we argue that access granularity should be adaptive and that the system should be dynamically optimized to meet the needs and characteristics of applications. Fine-grained accesses should be used judiciously, only when the overall tradeoffs indicate a positive gain. We demonstrate that adapting access granularity can be very effective with improvements in system throughput

This work is supported, in part, by the following organizations: The National Science Foundation under Grant #0954107, Intel Labs Academic Research Office for the Memory Hierarchy Innovations program, and The Texas Advanced Computing Center.

ISCA'11, June 4–8, 2011, San Jose, California, USA.



(a) Timing



(b) Effective throughput (useful data only)

Figure 1: Timing diagrams and effective throughput of several DRAM generations. 64-bit wide memory channels are assumed. Since DRAM core clock frequency stays constant at 200MHz, the ratio of I/O to core clock frequency doubles every generation of DDR DRAMs. Note that there are no 200MHz SDRAM products, but we present it here for comparison.

and power efficiency of more than 44% and 46%, respectively.

The remainder of this paper is organized as follows: Section 2 describes background and related work; Section 3 presents the proposed adaptive granularity memory system; the evaluation methodology is described in Section 4; the results and discussion is shown in Section 5; then, Section 6 concludes this paper.

2. BACKGROUND AND RELATED WORK

Before describing our adaptive-granularity architecture, we explain the tradeoffs with respect to access granularity in modern memory systems and discuss prior and related work.

2.1 Fine-Grained Cache Management

While orthogonal to our research on adaptive memory access granularity, work on cache architectures that support fine-grained data management is a necessary component of our design. These architectures provide mechanisms to maintain valid and dirty information at a granularity finer than just an entire cache line. The sector cache [28] was initially designed to reduce tag overheads by dividing each cache line into multiple sectors and providing each sector with its own dirty and valid bits. The decoupled sectored cache [40] and a pool-of-sectors cache [38] refine the sector design and enable more elastic mapping between sectors and tags to reduce the miss rate of a sector cache. The dual cache [18] uses two L1 caches, one with large line size and the other with small line size. Similarly the adaptive line size cache [43] gradually changes cache line size. Other techniques, spatial footprint prediction [25] and spatial pattern prediction [13], utilize a hardware predictor to reduce fetch traffic between L1 and L2 or to save leakage power by applying power-gating to the subblocks that are predicted to be non-referenced; and the line-distillation cache [36] splits the cache into a line-oriented cache and a word-oriented cache.

These techniques assume that off-chip memory can be accessed with fine granularity, which is unfortunately no longer true because modern DRAM systems evolved to provide high bandwidth with coarse access granularity. Our adaptive granularity memory system re-enables these fine-grained cache-management techniques. AGMS can be combined with any fine-grained cache-management technique, and in this paper we use a "sector cache" as a simple example (see Section 3).

2.2 Memory Access Granularity Tradeoffs

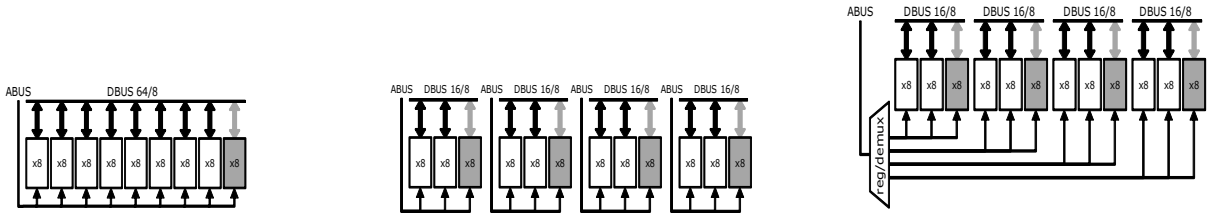
Modern memory systems do not provide truly-uniform random access. They are instead optimized for capacity first and for high bandwidth for sequential access second. In order to keep costs low, the DDRx interface relies on high spatial locality and is requiring an ever increasing minimum access granularity. Regardless of how much data is required by the processor, the off-chip DRAM system returns a certain minimum amount, often referred to as a *DRAM burst*. Therefore, in DDRx memory systems, the overall *minimum access granularity* is a product of the DRAM chip minimum burst length and the data width of a channel.

2.2.1 *Module and Interface Tradeoffs.*

Figure 1(a) shows simplified timing diagrams of data bus usage for several DRAM generations: single data rate SDRAM, DDR, DDR2, and DDR3. While the high density required from DRAM practically limits its core clock frequency to 200MHz, effective I/O data rates have increased up to 1600MHz for the latest DDR3. Note that newer DRAM generations transfer a larger chunk of data in the same time window (5ns in this example), doubling the peak bandwidth in each generation. This increase in bandwidth is achieved by employing n-bit prefetch and a burst access: n is 2 in DDR, 4 in DDR2, and 8 in DDR3. As a result, the minimum access granularity is increasing: 8B in SDRAM, 16B in DDR, 32B in DDR2, and 64B in DDR3 with a typical 64-bit wide channel. Thus, as DRAM technology advances, the relative cost of fine-grained accesses increases.

We carried out a simple experiment that compares the effective DRAM throughput of sequential accesses and fine-grained random accesses using DRAMsim [44]. We used `STREAM` [31] and `GUPS` [16] of the HPC Challenge Benchmarks [1] to represent sequential and random access patterns, respectively. As Figure 1(b) shows, near-peak bandwidth is easily achieved with sequential accesses (in `STREAM`), whereas fine-grained random accesses (in `GUPS`) have poor performance that barely improves over DRAM generations. Note that we provisioned enough banks and ranks (8 banks per rank and 8 ranks) so that bank-level parallelism can hide the penalty of the frequent bank conflicts of random accesses. Hence, the relatively low throughput of `GUPS` in DDR2 and DDR3 is primarily due to their large minimum access granularity. Only a fraction of the data transferred is actually used, lowering effective throughput.

A narrower data path can be used to decrease the minimum access granularity because the granularity is a prod-



(a) Conventional coarse-grain only (b) Many-narrow-channels (c) Sub-ranked

Figure 2: Comparison of a conventional memory system, a many-narrow-channels approach, and a sub-ranked memory system similar to MC-DIMM. The many-narrow-channels and sub-ranked in this figure provide 16B access granularity with DDR3. Gray boxes and arrows represent ECC storage and transfers, which increase significantly in (b) and (c) as a result of finer access granularity support. *ABUS* represents address/command bus and *DBUS X/Y* represents data bus, where X bits are for data and Y bits are for ECC.

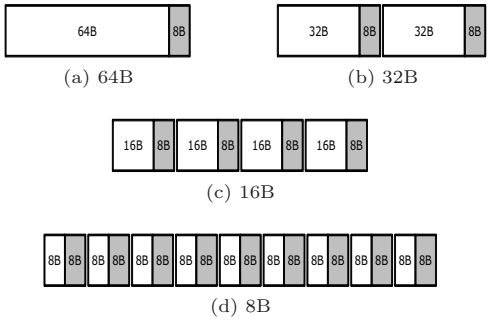
uct of minimum burst length and channel width. The burst length is dictated by the DRAM technology, but the channel width is a system design parameter. We can find such implementations in vector processors like the Cray Black Widow [6]: Its memory subsystem has many 32-bit wide DDR2 DRAM channels, providing 16B minimum access granularity. We refer to this style of memory subsystem as *many-narrow-channels*. Although the many-narrow-channels approach reduces minimum access granularity, such a design is inherently very expensive.

An alternative approach to this supercomputing-based many-narrow-channels design is to use DRAM sub-ranking. Recently, there have been multiple proposals for memory systems that can control individual DRAM devices within a rank: Rambus's micro-threading [45] and threaded memory module [46]; HP's MC-DIMM (multi-core dual in-line memory module) [8, 7]; Mini-rank memory system [51]; and Convey's S/G DIMM (scatter/gather dual in-line memory module) [10]. In this paper, we collectively refer to these techniques as *sub-ranked* memory systems. Figure 2 compares a conventional coarse-grain-only memory system, a many-narrow-channels approach, and a sub-ranked memory system similar to MC-DIMM.

Most sub-ranked memory proposals [46, 51, 8, 7] focus on energy efficiency of coarse-grained accesses by mitigating the "overfetch" problem. The ability of sub-ranked memory to support fine-grained accesses is briefly mentioned in [45, 10], but the tradeoffs are neither discussed nor evaluated. Unlike prior work on sub-ranked memory, AGMS provides efficient fine-grained access and dynamic reliability tradeoffs. Also, to the best of our knowledge we provide the first quantitative evaluation of sub-ranked memory systems for fine-grained access.

2.2.2 *Error Protection and Access Granularity.*

One important tradeoff that favors coarse-grained access is the efficiency of the error protection scheme. The high density of DRAM coupled with the large number of DRAM chips in many systems make memory one of the most susceptible components [39]. The most effective method to improve reliability is to tolerate errors using *error-checking and correcting* (ECC) codes [27]. With ECC, every access to memory is accompanied by an ECC operation to ensure that the access is correct. One pertinent characteristic of commonly used ECC is that its overhead grows sub-linearly with the size of the data it protects ($O(\log_2 n)$, where n is the size of the data). Therefore, the finer-grained the access, the larger the overhead of ECC. Moreover, since we cannot use an arbitrary width DRAM chip, the actual overhead in



(a) 64B (b) 32B (c) 16B (d) 8B

Figure 3: Tradeoff between access granularity and redundancy overheads. White and gray boxes represent data and ECC blocks respectively assuming a minimum access granularity of 8B (×8 DRAM chips and burst-8 accesses in DDR3). The finer the granularity, the higher the overhead of redundancy.

terms of chip count can be even larger. This ECC overhead manifests itself in reduced storage efficiency (more DRAM chips for the same data capacity) and lower effective pin bandwidth and power efficiency (more pins and watts for the same data bandwidth). For example, typical overhead with ECC DIMMs is 12.5% with 8 bits of ECC information for every 64 bits of data (or a 16 bits of ECC for 128 bits of data). In Cray's Black Widow [6], many-narrow-channels memory subsystem has 25% overhead: 7-bit SEC-DED (single bit-error correct and double bit-error detect) for each 32 bits of data, but the actual overhead is 8 bits when using ×8 DRAM chips. Furthermore, providing 8B access granularity with DDR3 using the many-narrow-channels approach requires 100% ECC overhead (a 5-bit ECC provides SEC-DED protection for 8-bit data, but we still need at least one ×8 DRAM chip to store ECC information, resulting in 8 overhead bits per 8-bit of data) as well as additional control overhead. Supporting ECC in a sub-ranked memory system is also very expensive: MC-DIMM requires a 37.5% or higher ECC overhead for chipkill-correct [7], for example. Figure 3 summarizes the tradeoff between access granularity and storage efficiency.

In summary, neither a conventional coarse-grained memory system nor a fine-grained memory system, including many-narrow-channels and sub-ranking schemes, can provide optimal throughput and efficiency. Optimizing the system for coarse-grained accesses sacrifices throughput when spatial locality is low, while tuning for fine-grained accesses makes the overheads of control and/or ECC significant. Modern computing systems, however, require all of these merits: high throughput both for coarse-grained and fine-grained ac-

cesses and high reliability and availability levels, and all of these at low power and storage overheads.

2.2.3 Other Related Work.

Our adaptive granularity memory system is closely related to the Impulse memory controller [49]; it uses a shadow address space to provide the illusion of contiguous data for non-unit stride or indexed gather/scatter accesses. The Impulse memory controller translates a shadow address to potentially multiple physical addresses, and then collects multiple fine-grained data blocks to form a dense coarse-grained data block, reducing traffic on the bus between the cache controller and the off-chip memory controller. Unfortunately, on chip memory controllers in recent architectures as well as ever-increasing memory access granularity neutralize Impulse's advantages. Moreover, it is unclear how to support both fine-grained memory accesses and ECC with Impulse.

Similar to the tradeoff between storage efficiency and fine-grained throughput presented in this work, RAID-Z implemented in the ZFS file system [4] uses different reliability schemes for stores with varying granularities. RAID-Z, however, has completely different management mechanisms and characteristics because it deals with the much larger data blocks of hard disk drives.

3. ADAPTIVE GRANULARITY MEMORY SYSTEM

We propose the adaptive granularity memory system (AGMS) that combines the best of fine-grained and coarse-grained accesses. AGMS uses a coarse-grained configuration for memory regions with high spatial locality and a fine-grained configuration for memory regions with low spatial locality. The proposed mechanism is a vertical solution that requires collaboration between several system levels: The application provides preferred granularity information (Section 3.1); the OS manages per-page access granularity by augmenting the virtual memory (VM) interface (Section 3.2); a sector cache manages fine-grained data in the cache hierarchy (Section 3.3); and a sub-ranked memory system and mixed-granularity memory scheduling provide efficient handling of multiple access granularities within off-chip memory system (Section 3.4 and Section 3.5). We also discuss the tradeoffs in making granularity decisions (Section 3.6).

3.1 Application Level Interface

As explained in the previous section, the adaptive granularity memory system requires different memory protection schemes for different access granularities. The degree of redundancy, and thus the memory layout, has to change (see Section 3.4 for details). Consequently, the processor cannot adapt the granularity independent of the software. The tuning/adaptation can be done either statically at compile time or dynamically by the OS (we discuss the dynamic case later). In the static approach, the programmer or an auto-tuner provides granularity information through a set of annotations, hints, compiler options, and defaults that associate a specific tolerance mechanism with every memory location.[1] We envision that the programmer will declare a preferred access granularity when memory is allocated. More accurately, we allow the programmer to override the default access granularity using annotations and compiler hints. In Fortran, programmer annotations can take the form of another array attribute; in C, we can add a parameter to `malloc`.

[1] Note that a physical memory location can only be protected using a single mechanism at any given time because the protection scheme and redundant information need to be checked and updated consistently.

3.2 OS Support

Granularity and protection schemes are applied when physical memory is allocated and are thus closely related to the virtual memory manager. We augment the virtual memory interface to allow software to specify the preferred access granularity for each page. The per-page access granularity is stored in a page table entry (PTE) when a page is allocated. This information is propagated through the memory hierarchy along with requests, miss status handling registers, and cache lines so that the memory controller can use the information to control DRAM channels. Since most architectures have *reserved* bits in a PTE, we can accommodate the preferred access granularity similar to per-page cache attributes in the ×86 ISA [22].

Because the OS manages both access granularity and virtual memory, it is possible to dynamically adapt the granularity without application knowledge. This would require hardware support for determining access granularity, such as the mechanisms proposed for fine-grained cache management [25] as well as the OS to copy (migrate) pages or change granularity when paging. We leave further exploration of this idea to future work and discuss it further in Section 3.6.

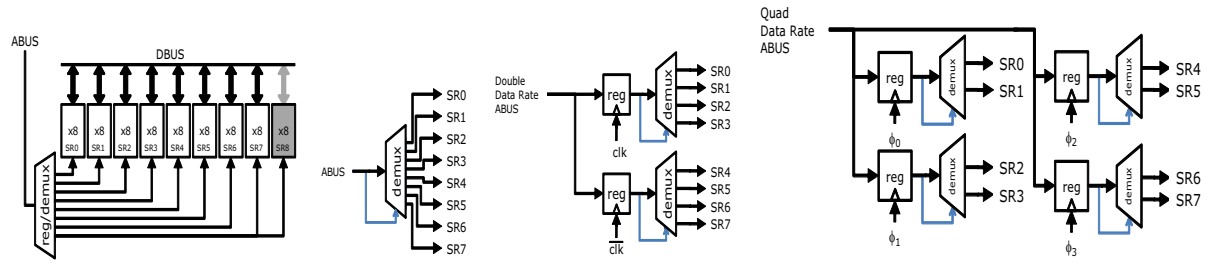
3.3 Cache Hierarchy

AGMS issues both coarse- and fine-grained requests to main memory and thus needs to manage both granularities of data within the cache hierarchy. The simplest way is to use a cache with a line size equal to the smallest access granularity, e.g. 8B. A better design choice is a sector cache [28]. A cache line in a sector cache is divided into multiple sectors, and each sector maintains its own valid and dirty bits, but there is only one tag for each multi-sectored cache line. Since sector caches do not increase address tag overhead, the additional cost is only the storage for valid and dirty bits: 14 bits per cache line when a 64B cache line is divided into eight 8B sectors.

While the more advanced cache architectures described in Section 2.1 can provide better management of fine-grained data, we choose a simple sector cache in this work to better isolate performance gains from adaptive granularity memory access; the simple sector cache allows a fair comparison between the adaptive granularity memory system and a conventional coarse-grain-only architecture.

3.4 Main Memory

Sub-Ranked Memory Systems. We leverage the sub-ranked memory system approach because it enables fine-grained accesses with minimal control overhead. We use a 64-bit wide memory channel with DDR3 DRAMs: eight ×8 DRAMs per rank. Figure 4(a) shows the main memory architecture for this study. The minimum access granularity in our system is 8B since we can control each memory device independently. Though sub-ranked memory systems can provide multiple granularities (8B, 16B, 24B, 32B, and 64B), we restrict access granularity to 64B (coarse-grained) and 8B (fine-grained) in this paper. A fine-grained 8B re-



(a) Sub-ranked memory system (b) 1× ABUS (c) 2× ABUS (d) 4× ABUS

Figure 4: Sub-ranked memory with a register/demux architecture of 1×, 2×, and 4× ABUS bandwidth. We use a register/demux similar to the one suggested in MC-DIMM [8]. Note that we do not use a register with the 1× ABUS configuration.

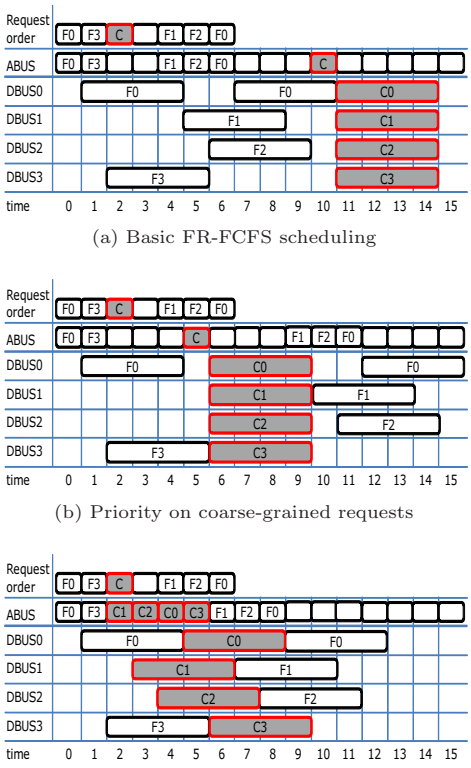
quest is serviced by a burst of 8 transfers from a single ×8 DRAM chip, and a coarse-grained 64B request is serviced by a burst of 8 transfers from 8 ×8 DRAM chips in a rank.

Address Bus Bandwidth. Because each fine-grained access reads or writes only 8B of data, it requires a factor of 8 more memory requests to achieve the equivalent throughput of coarse-grained accesses. In other words, fine-grained accesses will saturate an address/command bus (ABUS) that is designed for coarse-grained accesses, and greater command signaling bandwidth is required for fine-grained accesses to fully utilize the DRAM bandwidth. Many commercial high-end systems including Power 7 [24] and Cray Black Widow [6] as well as the custom address/command bus in the Convey S/G DIMM [10] and decoupled DIMM [52] already use or suggest 3 to 4 times, or even more, faster signaling for the ABUS. Increasing ABUS bandwidth is a matter of overall system optimization in terms of cost, power, and performance. We explore the design space of ABUS bandwidth later in this section.

Figure 4(b)-(d) show the architectures of the register/demux used in the sub-ranked memory system, and Figure 4(c) and Figure 4(d) show how 2× and 4× ABUS bandwidths are provided. While the 2× ABUS scheme can be achieved by using double data rate signaling as with the data bus (DBUS) with a relatively simple register/demux, the 4× ABUS scheme, which uses quad data rate signaling, may increase design complexity due to signal integrity issues.

Memory controllers. An interesting challenge in implementing adaptive granularity, compared to fixed granularity memory systems, is the effective co-scheduling of memory requests with different granularities; this includes buffering, scheduling, and providing quality of service. The example in Figure 5(a) illustrates how a coarse-grained request is unfairly deferred when there are many fine-grained requests, assuming the commonly used FR-FCFS [37] scheduling policy. The coarse-grained request at *time 2* cannot be immediately issued due to pending fine-grained requests F0 and F3, which were queued at *time 0* and *time 1* respectively. While the coarse-grained request waits in the queue, other fine-grained requests (F1, F2, and F0) arrive and are scheduled quickly since a fine-grained request can be serviced once its single associated sub-rank becomes ready. Coarse-grained requests, on the other hand, can only be scheduled when all sub-ranks are available at the same time. As a result, the coarse-grained request is serviced only after all fine-grained requests are serviced, potentially degrading performance significantly.

We propose two solutions to address this problem: (i) to give higher priority to coarse-grained requests; and (ii)



(a) Basic FR-FCFS scheduling

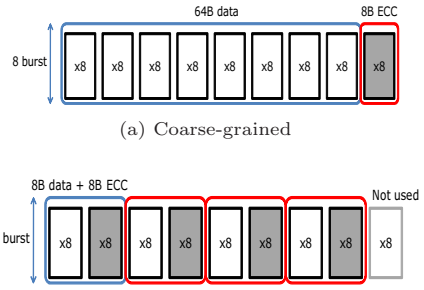
(b) Priority on coarse-grained requests

(c) Splitting coarse-grained requests

Figure 5: Scheduling examples. Fx represents a fine-grained request to sub-rank x, and C represents a coarse-grained request. There are four sub-ranks, and a memory device returns a 4-cycle burst after a request is made. FR-FCFS scheduling is used, and data bus contention is the only constraint in scheduling in this simple example.

to split a coarse-grained request into multiple fine-grained requests. The first solution prioritizes a coarse-grained request when its service is unfairly deferred due to fine-grained requests. Supporting different request priorities is a common feature of modern out-of-order memory schedulers. As shown in Figure 5(b) when the scheduler detects that a coarse-grained request is deferred due to fine-grained requests (at *time 2*), it raises the priority of the coarse-grained request. This prevents fine-grained requests with normal priority from being scheduled. As a result, the coarse-grained request finishes at *time 9*, after which fine-grained requests are serviced.

Our second solution splits a coarse-grained request into many fine-grained requests so that each fine-grained request (belonging to a single coarse-grained request) can be opportunistically scheduled. The original coarse-grained request



(a) Coarse-grained

(b) Fine-grained

Figure 6: Coarse-grained and fine-grained accesses with ECC

finishes when all its fine-grained requests are serviced. Figure 5(c) shows such an example. At *time 2*, the coarse-grained request is split into four fine-grained requests (C0, C1, C2, and C3), and the coarse-grained request finishes when all of them are serviced (at *time 9*). A caveat to this solution is that it potentially increases ABUS bandwidth requirement. We compare the two solutions in more detail later in this subsection.

3.4.1 Data layout.

When ECC is not used, the data layout in physical memory is the same for both fine-grained and coarse-grained pages. When ECC is enabled, on the other hand, we need to use a different data layout for fine-grained pages to account for the higher required redundancy. Figure 6 compares a coarse-grained and fine-grained accesses with ECC in a DDR3-based system. Fine-grained data can provide higher throughput with low spatial locality, but it increases the ECC overhead since every data block needs its own ECC code. The 8B minimum access granularity dictates at least 8B for ECC.

In this paper, we use a simple scheme in which a fine-grained physical page is twice as large as a coarse-grained nominal page, e.g. 8kB of storage for a 4kB data page. Each 8B of data is associated with 8B of ECC. Hence, a fine-grained request is serviced by accessing 16B in total. Memory controllers must interpret this change in addressing when making fine-grained data accesses with ECC, and the OS should manage physical memory pages accordingly. As a result, fine-grained pages have low storage efficiency, but can still provide better throughput than always using coarse-grained accesses. We store data and its associated ECC in different memory devices, providing better reliability than other embedded-ECC designs [51, 19, 15], which store ECC in the same DRAM row as the data it protects. A more generalized and flexible scheme such as Virtualized ECC [48] can manage ECC without changing physical data layout and can even handle granularities other than 64B and 8B easily, but we leave exploring such configurations to future work. We assume that ECC DIMMs are used, and ECC information for coarse-grained accesses is stored in the dedicated ECC DRAM chips.

3.5 AGMS Design Space

We now explore the design space of AGMS. We describe the details of the simulation settings and system parameters in Section 4. We use `GUPS` for exploring the design space because it is very memory-intensive and has many fine-grained requests. Furthermore, `GUPS` is often used as the gold standard for evaluating memory systems and network designs in large-scale systems and is considered to be a challenging benchmark [1]. We later show that AGMS provides significant benefits to real applications in addition to this important micro-benchmark.

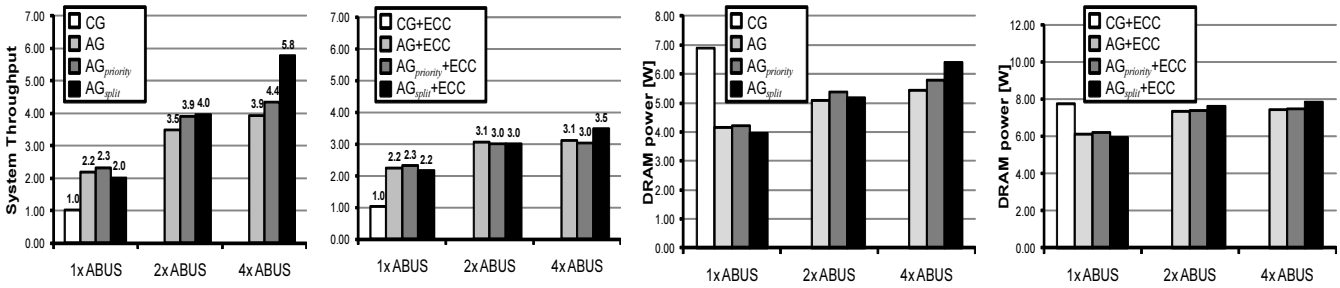
`GUPS` performs a collection of independent read-modify-write operations to random locations (8B elements in our experiments). `GUPS` has two buffers: an index array and a data array. The index array is accessed sequentially. The values stored in the index array are random numbers that are used for addressing the data array to be updated. For adaptive granularity, we define the index array as a coarse-grained region and the data array as a fine-grained region. We simulate a 4-core system with an instance of `GUPS` per core and a single 1067MHz DDR3 channel. We choose this relatively low-bandwidth configuration because it is representative of future systems that are expected to have larger compute to memory bandwidth ratios than current systems. Limited experiments with higher bandwidth yielded very similar results in the case of `GUPS`.

Figure 7(a)-(b) show the throughput of various system configurations using weighted speedup [17]. Compared to CG (coarse-grain-only), the AG schemes improve system throughput significantly: 100−130% with nominal 1× ABUS bandwidth, 150−200% with 2× ABUS, and up to 480% with 4× ABUS. Note that with AG, the index array uses coarse-grained accesses and the data array is accessed with fine granularity. The performance of the AG schemes with 1× and 2× ABUS is limited by ABUS bandwidth: They have almost 100% ABUS utilization, while data bus (DBUS) utilization is only 25% (1× ABUS) and 50% (2× ABUS). With 4× ABUS, DBUS utilization reaches more than 70%. Note that even with 1× ABUS, the effective throughput is still twice as high as that of a coarse-grain-only system.

The throughput of the baseline AG system is limited by coarse-grained requests to the index array that are unfairly deferred due to many fine-grained accesses. This effect is more pronounced with 2× or 4× ABUS. $AG_{priority}$ and AG_{split} overcome this inefficiency by allowing coarse-grained requests to complete in a timely manner. $AG_{priority}$ performs slightly better when ABUS bandwidth limits overall throughput (1× ABUS), but AG_{split} is best when ABUS bandwidth is higher (2× and 4× ABUS).

When ECC is enabled, the AG schemes improve performance by 120 − 130% with 1× ABUS, 200 − 210% with 2× ABUS, and up to 250% with 4× ABUS. One anomalous case is AG_{split} with 1× ABUS, where the throughput of AG_{split}+ECC is better than that of AG_{split}. The reason is that AG_{split}+ECC makes coarser requests because of the ECC data and thus requires less ABUS bandwidth. The coarse-grained requests are split into only four finer-grained requests (as opposed to eight requests with AG_{split} without ECC). Note that 4× ABUS does not provide any further improvements because the data bus is already heavily utilized (more than 70%) with 2× ABUS due to the extra bandwidth consumed by the redundant ECC information.

Figure 7(c) and Figure 7(d) compare the DRAM power consumption of the evaluated configurations. In general, the AG schemes consume much less power than CG because they avoid accesses and transfers of unnecessary data and mitigate DRAM "overfetch" [7]. Having higher ABUS bandwidth increases DRAM system utilization and DRAM power consumption. The significant improvements to sys-



(a) Weighted speedup (non-ECC) (b) Weighted speedup (ECC) (c) DRAM power (non-ECC) (d) DRAM power (ECC)

Figure 7: GUPS 4-core simulation for design space exploration. *CG* is a conventional fixed-granularity coarse-grained system; *AG* is the baseline adaptive granularity system with standard FR-FCFS scheduling; $AG_{priority}$ and AG_{split} are the adaptive granularity memory systems where the former uses higher priority for deferred coarse-grained requests and the latter splits coarse-grained requests into multiple fine-grained requests; The suffix *+ECC* represents ECC support in each configuration.

tem throughput, however, compensate for this increased power, leading to superior power efficiency.

Based on the evaluation results shown in Figure 7, in the rest of the paper, we use AG_{split} with 2× ABUS both for non-ECC and ECC configurations. Although 4× ABUS improves GUPS significantly, GUPS's access pattern of almost exclusively fine-grained requests is not common in real applications. Therefore, we choose a 2× ABUS configuration over a 4× ABUS system given its design complexity and power overhead of a 4× ABUS system. Both $AG_{priority}$ and AG_{split} perform equally well with 2× ABUS bandwidth in general, but AG_{split} can utilize bandwidth more efficiently in a few cases.

3.6 Access Granularity Tradeoffs

While fine-grained accesses can utilize off-chip bandwidth more efficiently than coarse-grained ones, we should also consider that fine-grained access may degrade performance because of potentially higher average memory access latency. Ideally, bottom-line performance can be used to determine granularity, but the high cost of copying (migrating) a page to change its data layout (when ECC is used) for a different access granularity dictates that such changes be infrequent. As a result, common approaches to "experiment" dynamically and monitor performance are ill-suited for granularity selection. Instead, we propose a heuristic that incorporates metrics of spatial locality and DRAM characteristics.

The first component of our heuristic is cache spatial locality. Fine-grained accesses avoid fetching unnecessary data and utilize scarce off-chip bandwidth resources more effectively. Minimizing traffic generally reduces memory power consumption and can maximize system throughput when memory throughput is the bottleneck. We estimate the spatial locality of a page by averaging the number of words used in each fetched cache line for that page. Thus, we can estimate the average traffic for fetching a line from a page as follows:

- The cost for a coarse-grained access (64B) is 73B: 64B data, 8B ECC, and 1B for control.
- The cost for a fine-grained access (8B) depends on the number of words referenced. For each referenced word, we need 17B: 8B data, 8B ECC, and 1B for control.

Consequently, fine-grained accesses minimize the amount of traffic if an average cache line in a page has fewer than 4 referenced words (for the configuration parameters used in this paper).

Minimizing traffic, however, does not always guarantee higher overall performance or efficiency. The reason is that with fine-grained requests, multiple accesses to the same line will all be treated as cache misses (they hit in the tag array, but miss on the actual data word because of the sector cache design). These misses could have been avoided with coarse-grained accesses. Thus, if memory bandwidth is not the bottleneck, fine-grained accesses can degrade performance. We account for this effect by considering DRAM access characteristics that make fetching additional data relatively cheap.

The second component of the heuristic is DRAM page hit rate. A high DRAM page hit rate reduces the potential gains of fine-grained data access since the relative overhead of fetching an entire cache line is smaller when page hit rate is high. When the page hit rate is low, fine-grained accesses allow more rapid transitions between pages and also increase the level of parallelism in the memory system (sub-rank level parallelism in addition to rank and bank level parallelism). Thus, if the page hit rate is high, pre-fetching extra data with coarse-grained accesses does not significantly impact effective memory throughput and can improve cache hit rate and average load latency.

We combine cache-line spatial locality and page hit rate by adding a penalty (α) to fine-grained accesses when page hit rate is high: We opt for fine granularity if $\alpha\times$ fine-grained-cost $\leq$ coarse-grained-cost. α is 1.0 when only one word is referenced for more than 70% of cache lines in a page. Otherwise, α is determined by DRAM page hit-rate: 1.0 if page hit-rate is lower than 60%; 1.3 if lower than 70%; 1.8 if lower than 80%; and 3.0 otherwise. These weights were chosen arbitrarily, and we determined that performance is insensitive to this choice to a large extent.

The two components of the granularity-decision heuristic can be calculated statically, using programmer hints or compiler analyses, or dynamically, by allowing the OS to track access characteristics. In this paper, we use an intermediate approach to identify fine-grained data regions using off-line profiling. Instead of modifying the application code and OS, we use the profiler to guide our decisions. We discuss the potential merits of this approach compared to dynamic prediction in the evaluation section, but overall, we determine that once an application starts running, its granularity characteristics are stable. Note that the OS can always dynamically override granularity decisions and set all pages to coarse-grained if it determines that to be the most beneficial approach.

Table 1: Benchmark statistics: Last-level cache (LLC) MPKI, DRAM page hit rates and IPC are estimated from the baseline cycle-accurate simulation with a single core; average number of used words per cache line is gathered using PIN.

Benchmark	Input size	IPC	LLC MPKI	DRAM page hit rate	Avg. referenced words per cache line
SPEC					
mcf	ref	0.24	31.3	19.1%	3.59
omnetpp	ref	0.49	11.6	47.8%	3.22
libquantum	ref	0.45	15.6	98.9%	4.09
bzip2	ref	0.80	3.2	57.1%	3.63
hmmer	ref	0.98	0.87	91.3%	7.93
astar	ref	0.87	0.59	44.0%	2.86
lbm	ref	0.56	22.9	82.6%	3.92
PARSEC					
canneal	400k elements	0.32	17.2	14.1%	1.87
streamcluster	16k points	0.49	14.5	86.8%	7.24
SPLASH2					
OCEAN	1026^2 grid	0.54	18.6	92.6%	6.68
Olden					
mst	2k nodes	0.12	41.6	40.5%	2.30
em3d	200k nodes	0.19	39.4	27.4%	2.62
HPCS					
SSCA2	64k nodes	0.27	25.4	25.5%	2.63
Micro-benchmarks					
Linked List	342 lists	0.04	111.9	34.2%	1.99
GUPS	8M elements	0.08	174.9	10.9%	1.84
STREAM	2M elements	0.4	51.9	96.5%	7.99

4. EVALUATION METHODOLOGY

We evaluate AGMS using a combination of PIN-based emulation [30] to collect granularity statistics for full application runs and detailed cycle-based simulations to determine impact on performance and power. For the cycle-based simulations, we use the Zesto simulator [29] integrated with a detailed DRAM simulator [23] that supports sub-ranked memory systems as well as variable ABUS bandwidth as described in Section 3.4. The DRAM simulator models memory controllers and DRAM modules faithfully, simulating buffering of requests, FR-FCFS [37] scheduling of DRAM commands, contention on shared resources such as ABUS and DBUS, and all latency and timing constraints of DDR3 DRAM. Banks and sub-ranks are XOR interleaved to minimize conflicts [50].

Workloads. We use a mix of several applications from SPEC CPU 2006 [42], PARSEC [9], Olden [12], SPLASH2 [47], and the HPCS [2] benchmark suites as well as micro-benchmarks such as GUPS [16], STREAM [31], and Linked List [3] (Table 1). We choose mostly memory-intensive applications without spatial locality, but also include applications that are not memory intensive and/or have high spatial locality. Note that with adaptive granularity, all applications can perform at least as well as on the baseline system because coarse-grained accesses are used by default, but we report numbers based on the heuristic described in Section 3.6.

For the cycle-based simulations, we ran a representative region from each application. We use Simpoint [20] to determine the regions for SPEC applications and manually skipped the initialization of the simpler and more regularly-behaved Olden, PARSEC, and SPLASH2 benchmark suites, HPCS SSCA2, and the micro-benchmarks. The size of each representative region is 200M instructions for the 4-core simulations and 100M instructions for the 8-core simulations. For the PIN-based experiments, we ran all applications to completion.

Table 2: Simulated base system parameters.

Processor core	4GHz x86 out-of-order core (4 or 8 cores)
L1 I-caches	32kB private, 2-cycle latency, 64B cache line
L1 D-caches	32kB private, 2-cycle latency, 64B cache line
L2 caches	256kB private for instruction and data 7-cycle latency, 64B cache line
Last-Level caches (LLC)	shared cache, 64B cache line 4MB 13-cycle latency for 4-core systems 8MB 17-cycles latency for 8-core systems 64B cache line
On-chip memory controller	FR-FCFS scheduler [37] 64-entry read queue, 64-entry write queue XOR-based bank, sub-rank mapping [50]
Main memory	1 72-bit wide DDR3-1066 channel 64-bit data and 8-bit ECC, ×8 DRAM chips 8 banks per rank, 4 ranks per channel parameters from Micron 1Gb DRAM [32]

Data page profiling. To collect profiling information, we use PIN [30] and emulate a 1-level 1MB 8-way set-associative cache with 64B lines. During program execution, we profile which 8B words are referenced within each cache line. We then aggregate the per-cache line cost across each 4kB page and determine the granularity recommendation that can be overridden dynamically by the OS (see Section 3.6), although we do not override the decision in this study. We use this static profiling method to report the ECC storage overheads of fine-grained pages in Section 5.1 and to identify fine-grained data pages for the cycle-based simulations (Section 5.2–5.3).

System Configurations. Table 2 gives the parameters of the baseline coarse-grain-only system used in our cycle-based simulations. The cache hierarchy of the base system has an instruction pointer prefetcher for the instruction caches and a stream prefetcher for the data caches.

We use the following configurations for evaluating the potential of the AGMS approach.

- CG+ECC: coarse-grain-only system as described in Table 2. ECC is stored in dedicated DRAMs.
- $\text{CG}_{\text{sub-ranked}}$+ECC: coarse-grain-only system, but it uses a sub-ranked memory system, similar to MC-DIMM [7]. We use the best configuration from [7], 4 sub-ranks per rank (see Figure 2(c)). For ECC support, we assume that each sub-rank has a dedicated ECC DRAM so that a 64B request is served by a burst of 32 transfers out of three ×8 DRAMs.
- FG+ECC: fine-grain-only system. The memory controller accesses only requested words (8B), but every fine-grained access is accompanied by 8B ECC; an 8B request is served by a burst of 8 transfers out of two ×8 DRAMs, of which one is for data and the other is for ECC. As a result, the effective memory channel is only 32 bits wide. Caches have 8 sectors per cache line.
- AG+ECC: AGMS with the AG_{split} scheme described in Section 3.4 and 2× ABUS. As FG+ECC, AG+ECC also uses sector caches in L1D and L2 caches and LLC.

In addition to the above configurations with ECC, we also evaluate systems without ECC: CG, $\text{CG}_{\text{sub-ranked}}$, FG, and AG. These non-ECC configurations are identical to their ECC counterparts except that they do not support ECC. CG and $\text{CG}_{\text{sub-ranked}}$ do not have dedicated ECC DRAMs. FG and AG (for fine-grained accesses) do not transfer ECC, yielding twice the peak fine-grained data rates of FG+ECC

Table 3: PIN-based data page profiling.

Benchmark	Total data	Fine-grained data	Fraction of fine-grained data
SPEC			
mcf	1676MB	1676MB	100%
omnetpp	175MB	159MB	91%
libquantum	122MB	0.1MB	0.1%
bzip2	208MB	124MB	59%
hmmer	64MB	0.1MB	0.2%
astar	349MB	258MB	74%
lbm	409MB	6.3MB	1.5%
PARSEC			
canneal	158MB	157MB	99%
streamcluster	9MB	0.1MB	1.6%
SPLASH2			
OCEAN	56MB	0.1MB	0.1%
Olden			
mst	201MB	193MB	96%
em3d	89MB	28MB	31%
HPCS			
SSCA2	186MB	18MB	10%
Micro-benchmarks			
Linked List	178MB	178MB	100%
GUPS	192MB	64MB	33%
STREAM	47MB	0.1MB	0.2%

and AG+ECC. Note that the AG schemes (AG and AG+ECC) can emulate the CG schemes (CG and CG+ECC) and the FG schemes (FG and FG+ECC) when we set all pages to coarse-grained or fine-grained, respectively.

Power models. We estimate DRAM power consumption using a power model developed by Micron Corporation [5]. For processor power analysis, we use the IPC-based mechanism presented in [7]: The maximum power per core is estimated as 16.8W based on a 32nm Xeon processor model using the McPAT 0.7 tool [26]; and half of the maximum power is assumed to be static (including leakage) with the other half being dynamic power that is proportional to IPC. In our experience, this rather simple measure of processor core power produces a power estimate that matches WATTCH [11] results well. Furthermore, our study focuses on main memory, and our mechanisms have minimal impact on the processor core's power behavior. We estimate LLC power using CACTI 6 [33]. To account for the cost of sector caches and the sub-ranked memory system with 2× ABUS and the register/demux, we add a 10% power penalty to the LLC and DRAM power consumption in AGMS, which we believe to be *very* conservative.

5. RESULTS AND DISCUSSION

This section presents our analysis of AGMS: Section 5.1 provides data page profiling results and reports the storage overhead of the redundant information in fine-grained pages; Section 5.2 and Section 5.3 present cycle-based simulation results from 4 cores and 8 cores, respectively.

5.1 Page Profiling Results

We use PIN with the simple cache model described in Section 4 to profile fine-grained pages as well as total data pages. The fraction of fine-grained data pages is important because the storage overhead of ECC for fine-grained pages is twice that of coarse-grained ones. As shown in Table 3, the fraction of fine-grained pages is low in `hmmer`, `libquantum`, `lbm`, `streamcluster`, `OCEAN`, `em3d`, `SSCA2`, `GUPS`, and `STREAM`, but is nearly 100% in many applications. Therefore, declaring a data page as fine grained must be done judiciously and used as part of overall system optimization (trading off performance and power-efficiency with a larger memory footprint).

Table 4: Application mix for 4-core simulations.

MIX-1	mcf	omnetpp	mcf	omnetpp
MIX-2	SSCA2	lbm	astar	SSCA2
MIX-3	libquantum	hmmer	mst	mcf
MIX-4	SSCA2	Linked List	mst	hmmer

For some applications, the choice is clear. As shown in the performance analysis in Section 5.2, those applications that have a small fraction of fine-grained pages perform very well with AGMS. Others still gain from using AGMS, but require significantly more memory capacity. This particular trade-off depends on many system-level parameters, and we leave this evaluation to future work.

5.2 4-core Cycle-Based Results

In this subsection, we present cycle-based simulation results of 4-core systems. Multi-programmed workloads are used for the 4-core system evaluation. We use 4 replicas of an application (suffix ×4) as well as application mixes (Table 4). We utilize weighted speedup [17] as the metric of system throughput. We use the fine-/coarse-grained decisions from our profiler. Both profiler and simulations used the same dataset, but the profiler was not heavily tuned.

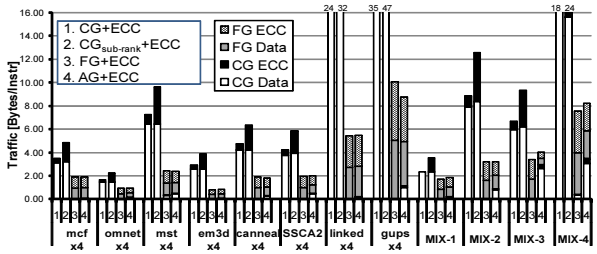
Off-Chip Traffic and Power. First, we compare the off-chip traffic and DRAM power consumption of CG+ECC, $CG_{sub\text{-}ranked}$+ECC, FG+ECC, and AG+ECC. Figure 8(a) shows the total memory traffic including ECC. AG+ECC reduces off-chip traffic by 66%, on average, compared to CG+ECC (56% excluding the micro-benchmarks: `GUPS`, `STREAM`, and `Linked List`).

The reduced off-chip traffic leads to lower DRAM power consumption as shown in Figure 8(b). Remember that we added a conservative 10% power penalty to the AG+ECC configurations. AG+ECC reduces DRAM power by 7% to 21% in most applications and 14% on average. DRAM power actually increases for `GUPS`, but that is a result of the much higher performance obtained; efficiency is significantly improved as discussed below.

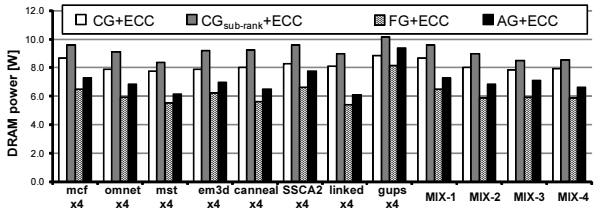
$CG_{sub\text{-}ranked}$+ECC, though the sub-ranked memory system was suggested for better energy efficiency, shows increased traffic and DRAM power consumption. This is mainly due to the cost of accessing redundant information; narrow access width in $CG_{sub\text{-}ranked}$+ECC necessitates high redundancy. FG+ECC, on the other hand, is effective in reducing traffic in most cases. FG+ECC, however, generates more traffic than AG+ECC. This is, again, due to ECC traffic; when spatial locality is high, coarse-grained accesses not only reduce miss rate but also minimize traffic including ECC. Though FG+ECC can minimize DRAM power consumption, since it touches only the necessary data, the reduced DRAM power consumption in FG+ECC does not necessarily lead to better performance or power efficiency as we show in the next paragraph.

Throughput and Power Efficiency. Figure 9(a) shows the system throughput of CG+ECC, $CG_{sub\text{-}ranked}$+ECC, FG+ECC, and AG+ECC. Overall, AG+ECC improves system throughput significantly: more than 130% in `GUPS`, 30% to 70% in `mst`, `em3d`, `SSCA2`, `canneal`, and `Linked List`, and 44% on average (22% on average excluding micro-benchmarks).

The results also show the advantage of adapting granularity compared to just using one of the mechanisms AGMS relies on fine-grained access (FG+ECC) and memory sub-ranking ($CG_{sub\text{-}ranked}$+ECC). Even in these applications that

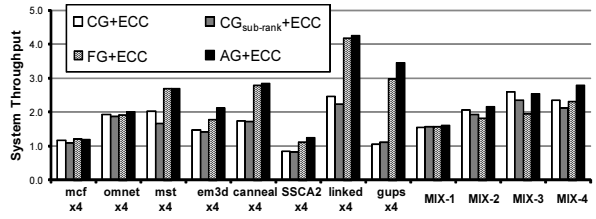


(a) Traffic

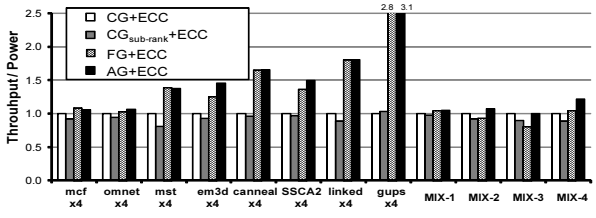


(b) DRAM Power

Figure 8: 4-core system off-chip traffic and power.



(a) System throughput



(b) Normalized system power efficiency

Figure 9: 4-core throughput and power efficiency.

can benefit from fine-grained access, AG+ECC consistently matches or outperforms FG+ECC.

We further evaluate applications with high spatial locality in Figure 10. AG+ECC, with most pages being coarse-grained, shows no degradation except `bzip2` relative to CG+ECC. FG+ECC, on the other hand, degrades system throughput significantly: 17% in `libquantum`, 34% in `bzip2`, 36% in `OCEAN`, 50% in `streamcluster`, and 48% in `STREAM`. `lbm` is the only case that FG+ECC improves throughput (7%). Though `lbm` has very high DRAM page hit rate (82.6%), `lbm` references fewer than 4 words per cache line on average; hence, the FG scheme (FG+ECC), with lower total traffic, can slightly improve performance.

We also show that sub-ranking alone (CG$_{\text{sub-ranked}}$+ECC) significantly impacts performance because access latency increases as it takes longer to transfer a coarse-grained cache line into the cache over a narrower channel. This effect is pronounced in our system configuration that has limited latency hiding with only a single thread per core.

MIX-3 and `bzip2` are the only experiments we ran in which performance degraded with the AG scheme, but the degradation is less than 4%. In `bzip2`, most memory accesses fall within coarse-grained regions so AG+ECC practically do not affect execution (degrade throughput by less than 4%). MIX-3 suffers a more apparent performance degradation. This is most likely because of unfair DRAM scheduling between the applications. The performance of `mcf` is degraded, but that of `mst` is improved. In the meanwhile, the performance of `libquantum` and `bzip2` results almost remained unchanged. This is because `mst` has a relatively high DRAM page hit rate compared to `mcf` (40.5% vs. 19.1%). Since the FR-FCFS scheduling policy used in our memory controller tries to maximize memory throughput, `mst`'s requests are favored over those of `mcf`, leading to unfair resource allocation between `mcf` and `mst`. We believe that combining the adaptive granularity scheme with a better scheduling mechanism that provides fairness, such as parallelism-aware batch scheduling [35], can overcome this inefficiency in MIX-3. Note that AGMS does improve the performance of MIX-D in the 8-core simulation (see Section 5.3). MIX-D has two instances of each application of MIX-3, and AG+ECC improves performance when relative off-chip bandwidth is more scarce.

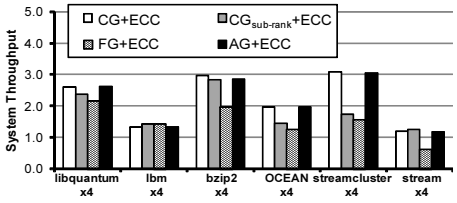
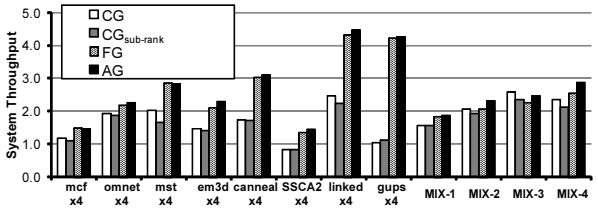
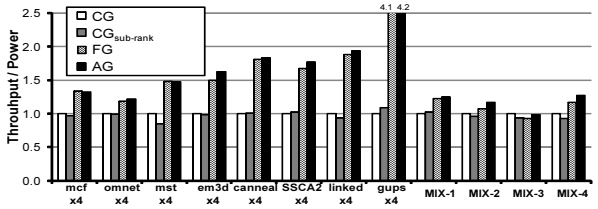


Figure 10: Applications with high spatial locality.



(a) System throughput



(b) Normalized system power efficiency

Figure 11: 4-core throughput and power efficiency (non-ECC).

Recall that AGMS allows every page to have its own granularity. Hence, we can nullify all performance degradation; the OS can override the granularity hint if it suspects an increase in unfairness. The OS can set fine-grained regions in a more conservative way or even use only coarse-grained accesses.

We report the system power efficiency in terms of throughput per unit power (including cores, caches, and DRAM) in Figure 9(b). With reduced DRAM power consumption, AG+ECC improves the system power efficiency except in MIX-3. AG+ECC improves efficiency by 46% on average (24% excluding micro-benchmarks). The AG scheme degrades the throughput per power of MIX-3 by only less than 2%, which is due entirely to the 10% DRAM power penalty

Table 5: Application mix for 8-core simulations.

MIX-A	mcf ×4	omnetpp ×4
MIX-B	SSCA2 ×2 omnetpp ×2	mcf ×2 mst ×2
MIX-C	SSCA2 mcf astar hmmer	omnetpp mst lbm bzip2
MIX-D	libquantum ×2 mst ×2	hmmer ×2 mcf ×2

we conservatively added to account for the register/demux. In many current systems that are coarse-grained only, however, registered DIMMs are already used for higher capacity. Compared to such systems, we do not expect any degradation in efficiency.

Note that we use power efficiency rather than a metric such as energy-delay product (EDP) because of our multi-programmed simulation methodology. All applications continue to run until the slowest-running application completes, and thus the amount of work varies from one configuration to another. Keep in mind that this is only for measuring power consumption, and the statistics for a specific core freeze when a core executes a fixed number of instructions so that the IPC comparison is based on the same number of instructions across different configurations. We believe our approach is the most appropriate for our analysis. Alternative approaches, such as FIESTA [21], propose that in each experiment the same number of instructions is executed. Thus, the amount of work does not vary across experiments, allowing energy comparison.

We also present the throughput and the overall power efficiency of non-ECC configurations in Figure 11. While overall tendency is the same as the results of ECC configurations, AG shows even further gains: 61% in throughput and 67% in power efficiency on average (34% and 40% excluding micro-benchmarks).

Note that we use the same profiler designed for AG+ECC, showing that AGMS is not sensitive to the profiler designs; using sub-optimal profiler in AG still provides significant gains in most applications. Furthermore, based on the improvements in FG and FG+ECC, simple per-thread decision (either all coarse-grained or all fine-grained) in the AGMS will lead to better performance and efficiency than coarse-grain-only baseline.

5.3 8-core Cycle-Based Results

This section presents results from 8-core system simulations. Systems with more cores are likely to have more memory channels, but we evaluate an 8-core system with a single channel. We chose this configuration because we expect systems in the future to generally have a lower memory bandwidth versus overall core arithmetic throughput. We use 8 replicas of an application (suffix ×8) as well as application mixes (Table 5).

The results of 8-core simulations (Figure 12) show similar trends to those of 4-core systems. The performance and power gains, however, are higher in the more bandwidth-constrained 8-core configurations. Performance is improved by 85% (59% excluding micro-benchmarks) with AG+ECC. Though we do not show, AG (without ECC) shows further gains of 116% throughput improvement (87% excluding micro-benchmarks). Reductions in DRAM power consumption and traffic are similar to those in the 4-core experiments. In future systems, where off-chip bandwidth will be severely limited, it is likely that virtually all applications will be bandwidth-bound; hence, we expect AGMS, which utilizes off-chip bandwidth more efficiently, to be even more beneficial.

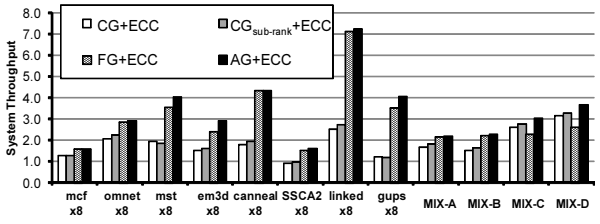


Figure 12: 8-core system throughput.

6. CONCLUSIONS AND FUTURE WORK

We present a novel architecture that enables a tradeoff between storage efficiency and fine-grained throughput and power efficiency. The adaptive granularity memory system (AGMS) utilizes finite off-chip bandwidth more efficiently: Fine-grained memory access minimizes off-chip traffic and reduces DRAM power consumption by not transferring unnecessary data, while also increasing concurrency in memory accesses and improving performance; coarse-grained memory access minimizes control and redundancy overheads and can potentially reduce miss rate. In 4-core systems, AGMS, even with higher ECC overhead, improves system throughput and power efficiency by 44% and 46% respectively. It also reduces memory power consumption and traffic by 14% and 66%. AGMS, though we design it for ECC-enabled memory systems, is also beneficial for non-ECC memory systems and provides further gains: Throughput improvement is 61%; and power efficiency gain is 67%. When off-chip bandwidth is more constrained, as in our 8-core configurations, AGMS is even more beneficial and important: System throughput increases are 85% with ECC and 116% without ECC. Thus, we conclude that adapting access granularity will be more effective in future systems, where off-chip bandwidth is relatively scarce. Note that these promising results were obtained with a very rough profiler for determining the preferred granularity. We expect to improve on this work by developing more sophisticated heuristics, utilizing more programmer knowledge, and studying dynamic adaptivity.

We will also continue to improve AGMS by studying more efficient memory controller designs when multiple access granularities are present, supporting other access granularities such as 16B or 32B in a more generalized framework, and exploring stronger protection mechanisms, e.g. chipkill-correct. Another direction is to apply the adaptive granularity scheme to different architectures: GPUs, vector or stream architectures, or general purpose processors with vector extensions. These architectures have explicit gather/scatter operations, requiring efficient fine-grained memory accesses. Hence, the adaptive granularity memory system will present more benefits in such architectures.

7. REFERENCES

[1] HPC challenge. http://icl.cs.utk.edu/hpcc/hpcc_results.cgi.

[2] HPCS scalable synthetic compact application (SSCA). http://www.highproductivity.org/SSCABmks.htm.

[3] Linked list traversal micro-benchmark. http://www-sal.cs.uiuc.edu/~zilles/llubenchmark.html.

[4] ZFS the last word in file systems. http://www.opensolaris.org/os/community/zfs/docs/zfs_last.pdf.

[5] Calculating memory system power for DDR3. Technical Report TN-41-01, Micron Technology, 2007.

[6] D. Abts, A. Bataineh, S. Scott, G. Faanes, J. Schwarzmeier, E. Lundberg, M. Byte, and G. Schwoerer. The Cray Black Widow: A highly scalable vector multiprocessor. In *Proc. the Int'l Conf. High Performance Computing, Networking, Storage, and Analysis (SC)*, Nov. 2007.

[7] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber. Future scaling of processor-memmory interfaces. In *Proc. the Int'l Conf. High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2009.

[8] J. H. Ahn, J. Leverich, R. Schreiber, and N. P. Jouppi. Multicore DIMM: An energy efficient memory module with independently controlled DRAMs. *IEEE Computer Architecture Letters*, 8(1):5–8, Jan. - Jun. 2009.

[9] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. Technical Report TR-811-08, Princeton Univ., Jan. 2008.

[10] T. M. Brewer. Instruction set innovations for the Convey HC-1 computer. *IEEE Micro*, 30(2):70–79, 2010.

[11] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proc. the 27th Ann. Int'l Symp. Computer Architecure (ISCA)*, Jun. 2000.

[12] M. C. Carlisle and A. Rogers. Software caching and computation migration in Olden. Technical Report TR-483-95, Princeton University, 1995.

[13] C. Chen, S.-H. Yang, B. Falsafi, and A. Moshovos. Accurate and complexity-effective spatial pattern prediction. In *Proc. the 10th Int'l Symp. High-Performance Computer Architecture (HPCA)*, Feb. 2004.

[14] C. L. Chen and M. Y. Hsiao. Error-correcting codes for semiconductor memory applications: A state-of-the-art review. *IBM J. Research and Development*, 28(2):124–134, Mar. 1984.

[15] R. Danilak. Transparent error correction code memory system and method. US Patent, US 7,117,421, Oct. 2006.

[16] Earl Joseph II. GUPS (giga-updates per second) benchmark. `http://www.dgate.org/~brg/files/dis/gups/`.

[17] S. Eyerman and L. Eeckhout. System-level performance metrics for multiprogram workloads. *IEEE Micro*, 28(3):42–53, 2008.

[18] A. Gonzalez, C. Aliagas, and M. Valero. A data cache with multiple caching strategies tuned to different types of locality. In *Proc. the Int'l Conf. Supercomputing (ICS)*, Jul. 1995.

[19] M. J. Haertel, R. S. Polzin, A. Kocev, and M. B. Steinman. ECC implementation in non-ECC components. US Patent Pending, Serial No. 725,922, Sep. 2008.

[20] G. Hamerly, E. Perelman, J. Lau, and B. Calder. SimPoint 3.0: Faster and more flexible program analysis. In *Proc. the Workshop on Modeling, Benchmarking and Simulation (MoBS)*, Jun. 2005.

[21] A. Hilton, N. Eswaran, and A. Roth. FIESTA: A sample-balanced muilti-program workload methodology. In *Proc. the Workshop on Modeling, Benchmarking and Simulation (MoBS)*, Jun. 2009.

[22] Intel Corp. *Intel® IA-64 and IA-32 Architecture Software Developer's Manual*, Mar. 2010.

[23] M. K. Jeong, D. H. Yoon, and M. Erez. DrSim: A platform for flexible DRAM system research. `http://lph.ece.utexas.edu/public/DrSim`.

[24] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd. Power7: IBM's next-generation server processor. *IEEE Micro*, 30(2):7–15, 2010.

[25] S. Kumar and C. Wilkerson. Exploiting spatial locality in data caches using spatial footprints. In *Proc. the 25th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 1998.

[26] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proc. the 42nd Ann. IEEE/ACM Int'l Symp Microarchitecture (MICRO)*, Dec. 2009.

[27] S. Lin and D. J. Costello Jr. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1983.

[28] J. S. Liptay. Structural aspects of the system/360 model 85, part II: The cache. *IBM Systems Journal*, 7:15–21, 1968.

[29] G. H. Loh, S. Subramaniam, and Y. Xie. Zesto: A cycle-level simulator for highly detailed microarchitecture exploration. In *Proc. the Int'l Symp. Performance Analysis of Software and Systems (ISPASS)*, Apr. 2009.

[30] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. PIN: Building customized program analysis tools with dynamic instrumentation. In *Proc. the ACM Conf. Programming Language Design and Implementation (PLDI)*, Jun. 2005.

[31] J. D. McCalpin. STREAM: Sustainable memory bandwidth in high performance computers. `http://www.cs.virginia.edu/stream/`.

[32] Micron Corp. *Micron 1 Gb ×4, ×8, ×16, DDR3 SDRAM: MT41J256M4, MT41J128M8, and MT41J64M16*, 2006.

[33] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. CACTI 6.0. Technical report, HP Labs., Apr. 2009.

[34] R. C. Murphy and P. M. Kogge. On the memory access patterns of supercoputer applications: Benchmark selection and its implications. *IEEE Transactions on Computers*, 56(7):937–945, Jul. 2007.

[35] O. Mutlu and T. Moscibroda. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In *Proc. the 35h Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2008.

[36] M. K. Qureshi, M. A. Suleman, and Y. N. Patt. Line distillation: Increasing cache capacity by filtering unused words in cache lines. In *Proc. the 13th Int'l Symp. High Performance Computer Architecture (HPCA)*, Feb. 2007.

[37] S. Rixner, W. J. Dally, U. J. Kapasi, P. R. Mattson, and J. D. Owens. Memory access scheduling. In *Proc. the 27th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2000.

[38] J. B. Rothman and A. J. Smith. The pool of subsectors cache design. In *Proc. the 13th Int'l Conf. Supercomputing (ICS)*, Jun. 1999.

[39] B. Schroeder, E. Pinheiro, and W.-D. Weber. DRAM errors in the wild: A large-scale field study. In *Proc. the 11th Int'l Joint Conf. Measurement and Modeling of Computer Systems (SIGMETRICS)*, Jun. 2009.

[40] A. Seznec. Decoupled sectored caches: Conciliating low tag implementation cost. In *Proc. the 21st Ann. Int'l Symp. Computer Architecture (ISCA)*, Apr. 1994.

[41] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos. Spatial memory streaming. In *Proc. the 33rd Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2006.

[42] Standard Performance Evaluation Corporation. SPEC CPU 2006. `http://www.spec.org/cpu2006/`, 2006.

[43] A. V. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, and X. Ji. Adapting cache line size to application behavior. In *Proc. the Int'l Conf. Supercomputing (ICS)*, Jun. 1999.

[44] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob. DRAMsim: A memory-system simulator. *SIGARCH Computer Architecture News (CAN)*, 33:100–107, Sep. 2005.

[45] F. A. Ware and C. Hampel. Micro-threaded row and column operations in a DRAM core. In *Proc. the first Workshop on Unique Chips and Systems (UCAS)*, Mar. 2005.

[46] F. A. Ware and C. Hampel. Improving power and data efficiency with threaded memory modules. In *Proc. the Int'l Conf. Computer Design (ICCD)*, 2006.

[47] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proc. the 22nd Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 1995.

[48] D. H. Yoon and M. Erez. Virtualized and flexible ECC for main memory. In *Proc. the 15th Int'l. Conf. Architectural Support for Programming Language and Operating Systems (ASPLOS)*, Mar. 2010.

[49] L. Zhang, Z. Fang, M. Parker, B. Mathew, L. Schaelicke, J. Carter, W. Hsieh, and S. McKee. The Impulse memory controller. *IEEE Transactions on Computers, Special Issue on Advances in High Performance Memory Systems*, 50(11):1117–1132, Nov. 2001.

[50] Z. Zhang, Z. Zhu, and X. Zhang. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *Proc. the 33rd IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, Dec. 2000.

[51] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu. Mini-rank: Adaptive DRAM architecture for improving memory power efficiency. In *Proc. the 41st IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, Nov. 2008.

[52] H. Zheng, J. Lin, Z. Zhang, and Z. Zhu. Decoupled DIMM: Building high-bandwidth memory systems using low-speed DRAM devices. In *Proc. the 36th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2009.